EXHIBIT 135

Hive - A Petabyte Scale Data Warehouse...

Engineering at Meta · Last edited March 12, 2021 · 11 minute read

Hive - A Petabyte Scale Data Warehouse using Hadoop

A number of engineers from Facebook are speaking at the Yahoo! Hadoop Summit today about the ways we are using Hadoop and Hive for analytics. Hive is an open source, peta-byte scale date warehousing framework based on Hadoop that was developed by the Data Infrastructure Team at Facebook. In this blogpost we'll talk more about Hive, how it has been used at Facebook and its unique architecture and capabilities.

Scalable analysis on large data sets has been core to the functions of a number of teams at Facebook - both engineering and non-engineering. Apart from ad hoc analysis and business intelligence applications used by analysts across the company, a number of Facebook products are also based on analytics. These products range from simple reporting applications like Insights for the Facebook Ad Network, to more advanced kind such as Facebook's Lexicon product. As a result a flexible infrastructure that caters to the needs of these diverse applications and users and that also scales up in a cost effective manner with the ever increasing amounts of data being generated on Facebook, is critical. Hive and Hadoop are the technologies that we have used to address these requirements at Facebook.

When we started at Facebook in 2007 all of the data processing infrastructure was built around a data warehouse built using a commercial RDBMS. The data that we were generating was growing very fast - as an example we grew from a 15TB data set in 2007 to a 2PB data set today. The infrastructure at that time was so inadequate that some daily data processing jobs were taking more than a day to process and the situation was just getting worse with every passing day. We had an urgent need for infrastructure that could scale along with our data and it was at that time we then started exploring Hadoop as a way to address our scaling needs. The fact that Hadoop was an open source project that was already used at petabyte scale and provided scalability using commodity hardware was a very compelling proposition for us. Moreover, the same jobs tha
taken more than a day to complete could now be completed within a few hours using Hadoo

counts or averages. Hadoop lacked the expressibility of popular query languages like SQL and as a result users ended up spending hours (if not days) to write programs for typical analysis. It was very clear to us that in order to really empower the company to analyze this data more productively, we had to improve the query capabilities of Hadoop. Bringing this data closer to users is what inspired us to build Hive. Our vision was to bring the familiar concepts of tables, columns, partitions and a subset of SQL to the unstructured world of Hadoop, while still maintaining the extensibility and flexibility that Hadoop enjoyed. Hive was open sourced in August 2008 and since then has been used and explored by a number of Hadoop users for their data processing needs.

Right from the start, Hive was very popular with all the users internally at Facebook. Today we regularly run thousands of jobs on the cluster with hundreds of users using this system for a wide variety of applications. Hive/Hadoop cluster at Facebook stores more than 2PB of uncompressed data and routinely loads 15 TB of data daily. It is heavily used for simple summarization jobs, business intelligence and machine learning and many other applications.

In the following sections we provide more details about Hive architecture and capabilities.

Data Model

Hive organizes data in tables and partitions. As an example, advertisement impressions can be stored in an ad_impressions table which is partitioned by date, e.g. 2009-05-01 partition for May 1, 2009 data and 2009-04-31 for April 31, 2009 data. The data for a particular date goes into a partition for that date. A good partitioning scheme allows Hive to prune data while processing a query and that has a direct impact on how fast a result of the query can be produced, e.g. queries on the impressions for a single day do not have to process data for other days. A good partitioning scheme also helps in managing the data and opens up possibilities to archiving older data to cheaper storage while keeping the more recent data on the main cluster.

Behind the scenes, Hive stores partitions and tables into directories in Hadoop File System (HDFS). In the previous example the table ad_impressions could be mapped to /warehouse /ad_impressions while each of the partitions can be mapped to /warehouse/ad_impressions /ds=2009-05-01 and /warehouse/ad_impressions/ds=2009-04-31 where ds (date stamp) is a partitioning column. The partitioning scheme can have multiple columns as well in which case each partitioning column maps to a level within the directory name space. Within each partition the data can be bucketed into individual files. The bucketing can be random or hashed on a par column. Setting up a good bucketing scheme helps in fast sampling of the data which can b d for fast analysis and data verification on sampled data.

floats and strings, or can be complex types such as maps, lists and structures. Moreover, Hive can store and query arbitrary user defined types through its extensibility mechanisms as described later in this post.

System Architecture and Components

Hive comprises of the following major components:

* Metastore: To store the meta data.
* Query compiler and execution engine: To convert SQL queries to a sequence of map/reduce jobs that are then executed on Hadoop.
* SerDe and ObjectInspectors: Programmable interfaces and implementations of common data formats and types.
* UDF and UDAF: Programmable interfaces and implementations for user defined functions (scalar and aggregate functions).
* Clients: Command line client similar to Mysql command line and a web UI.

The metastore stores the information about the tables, partitions, the columns within the tables, the SerDes used to serialize and deserialize from partitions, the locations of the data for these units within hdfs, bucketization attributes etc. It also provides a basic key value store which can be used to store any kind of meta data with these objects.

The Query compiler uses the information stored in the metastore to convert SQL queries into a sequence of map/reduce jobs, e.g. the following query

```
SELECT * FROM t where t.c = 'xyz'
```

is converted into a single map only job while a query of the form

```
SELECT t1.c2 FROM t1 JOIN t2 ON (t1.c1 = t2.c1)
```

is converted into a single map/reduce job that computes the join.

More complex plans are also possible, e.g. in some instances a group by of the form

```
SELECT t1.c1, count(1) from t1 group by t1.c1
```

may be converted into 2 map/reduce jobs in case there is skew in the data on c1. The first

aggregation.

Hive also supports more advanced query constructs like FROM clause sub queries and UNION ALL. The compiler also creates smart plans through a number of optimizations such as:

* Predicate pushdown
* Combining multiple map/reduce jobs to single map/reduce jobs
* Column pruning
* Map-side aggregations
* Map-side joins

Once the plan is generated the compiler passes it to the execution engine that executes the map/reduce jobs, launching them to the Hadoop cluster in the order of their dependencies.

Together all these components work to provide a powerful framework and language to manage and query data. Apart from the query components the language framework also provides normal DDL capabilities like creating, deleting and modifying tables and partitions and describing and showing tables as well.

Data Formats, User Defined Types and User Defined Functions

Hive is flexible enough to parse different data formats and data types through its SerDe and ObjectInspector Java interfaces. It provides some built in implementations of these interfaces for commonly occurring data formats such as delimited text files, csv files and some efficient binary formats. However, if the data is not in any of these formats (e.g. it may be XML data or in some proprietary format that is used by other applications), it is easy to write a conforming implementation class of the SerDe interface, drop it into a jar at a known location and instruct Hive to use the implemented class to serialize and deserialize the data stored in that particular partition.

The same kind of flexibility is available for rich data sets. With the built in implementations of the ObjectInspector Java interface, it is fairly easy to come up with ObjectInspectors for User Defined Data Types, e.g. a Point data type that comprises of two floating point numbers, can be programed as a PointObjectInspector class that implements how this data type can be represented in Hive's processing engine. Additionally, the user can program and register their own User Defined Functions to work on such data types and use them in SQL, e.g. on the PointObjectInspector, the user can define functions like get_X and get_Y to return the x and y co-ordinates of a point a
then can use these functions in a query such as

where the points_table has a single point column which is of the PointObjectInspector type and is stored in a file using the PointSerDe.

The ability to program and register custom functions (both simple functions and aggregate functions) also allows the users to extend Hive in areas where functionality may not have been provided out of the box. It is fairly easy to code up such function implementations using the UDF(User Defined Functions) and UDAF(User Defined Aggregate Functions) Java interfaces, package these implementations into a jar at a known location and instruct Hive to pick those classes up to be used in SQL statements, e.g while a today() function - that gives the current date - is not provided by Hive out of the box, it is very easy to code this up and register it with Hive so that it can be used in the queries. This is shown in the following example.

```
 CREATE TEMPORARY FUNCTION today AS 'yourpackage.UdfTodayImplementation';
SELECT * FROM test_tab T where T.dt < today();
```

All these abilities make Hive a very flexible and extensible system when it comes to supporting different data formats, data types or functions.

Extensibility and Writing Your Own Map/Reduce

Hive also provides a lot of flexibility in supporting computations that may not completely fit the SQL processing model. Machine learning model generation, building different kinds of indexes and many other applications are not easily expressible in SQL and Hive is able to support such applications by providing language constructs that allow the user to plug in their own transformation scripts in an SQL statement. This is done through the MAP, REDUCE, TRANSFORM, DISTRIBUTE BY, SORT BY and CLUSTER BY keywords in the SQL extensions that the Hive Query Language supports. As an example, the canonical word count example on documents stored in a table can be programmed using map/reduce and Hive as follows:

```
CREATE TABLE docs(contents STRING);
FROM

   (MAP docs.contents USING 'tokenizer_script' AS word, cnt
           FROM docs
           CLUSTER BY word) map_output

REDUCE map_output.word, map_output.cnt USING 'count_script' AS word, cnt;
```

word that it finds in contents, the count_script adds up all the 1s for each word to emit out a word count. The CLUSTER BY word clause in the statement tells Hive to distribute the map output to the reducers by hashing on word.

Such flexibility allows Hive to be used for non SQL kind of analysis as well and has been extensively used at Facebook for a variety of such tasks.

Interoperability

A hallmark of a successful system is also how it inter operates with other tools and systems. In order to address that need we have built a JDBC driver (and are currently also building an ODBC driver) for Hive. Though these are still experimental but with these drivers in place Hive can be made to inter operate with other analytics tools and repositories in a standards compliant manner. With the ODBC driver, Hive can be made to talk to BI engines like Micro strategy thereby further exposing the power of data and analytics to users through tools that they are familiar with.

Conclusion

As an open source project Hive has had a number of external contributions as well – including an open source web UI and a Columnar storage format to name a few. We continue to look for contributions from the community and are committed to building a strong community around Hive. The Hive wiki in apache is available at http://wiki.apache.org/hadoop/Hive and contains a lot of information about the Hive open source community and more details about the project.

Over the last one and a half years Facebook's data team has built a number of other systems in order to make scalable data processing and analytics easier. While Hive is open source, the other technologies are not yet available in the public domain but they have played an equally important role in making analytics on large data sets a reality for our users. These include a UI for Hive called Hipal that is used extensively within Facebook for ad hoc analysis, Databee that is used extensively to manage dependencies between different data processing tasks and some data discovery and scheduling tools. Together these technologies combine to provide powerful tools for users to analyze data and build analytics based applications on top of this data.

Ashish Thusoo and Joydeep Sen Sarma are on the Data Infrastructure team at Facebook and started the Hive project at Facebook.